

GCStack+ and GCScaler+: Closing the Calibration Gap in GPU Performance Modelling for Modern Microarchitectures

Tim Lühnen*, Ulf Kulau*, Devashree Tripathy[†] and Sohan Lal[‡]

*Hamburg University of Technology, Germany

[†]Indian Institute of Technology Bhubaneswar, India

[‡]Indian Institute of Technology Jammu, India

*{tim.luehnen, ulf.kulau}@tuhh.de

[†]devashreetripathy@iitbbs.ac.in

[‡]sohan.lal@iitjammu.ac.in

Abstract—GPU analytical performance models enable rapid design space exploration, but their accuracy depends heavily on the cycle-level simulators used for calibration. Existing tools such as GCStack and GCScaler are calibrated using Accel-Sim, whose execution pipeline is based on the NVIDIA Tesla architecture from 2006, featuring operand collector units, hardware scoreboards, and a Greedy-Then-Oldest warp scheduler. Recent reverse engineering studies have shown that modern NVIDIA GPUs eliminate operand collectors in favor of compiler-guided dependency management and adopt a Greedy-Then-Youngest warp scheduling policy. Although the simulator can still produce reasonable overall cycle counts, it often misattributes the underlying stalls in its CPI stacks and cache statistics. These inaccuracies are then passed onto any analytical model calibrated using them. In this paper, we revisit the calibration process used by analytical GPU performance models. We develop GCStack+, which generates CPI stacks and cache statistics using a simulator that reflects modern NVIDIA GPU microarchitecture, and GCScaler+, which uses these calibration inputs for performance prediction. The framework also adds support for Blackwell-generation instruction set, a protobuf-based trace format, and compiler-guided warp scheduling. Validated against an NVIDIA RTX 5070 (Blackwell) across nine Rodinia benchmarks, GCScaler+ reduces the geometric mean cycle prediction error from 18.2% (GCScaler) to 12.1%, at a 10.3× faster runtime than the simulator used during calibration.

Index Terms—CUDA, GPU, Analytical Modeling

I. INTRODUCTION

Graphics Processing Units (GPUs) have become the dominant accelerator platform for High Performance Computing (HPC) workloads spanning machine learning [1]–[3], scientific simulation [4], [5], and large-scale data analytics. As GPU architectures evolve rapidly across generations, GPU architects depend on performance analysis tools to identify bottlenecks in current designs and to explore the design space for next-generation hardware. These tools broadly fall into three categories. Cycle-level timing simulators [6]–[8] model the microarchitecture in detail but require hours to days per design point. Analytical models [9]–[11] abstract the architecture into closed-form expressions for rapid evaluation. Hybrid approaches such as GCStack combined with GCScaler [12], [13] collect fine-grained cycle stacks from a single baseline

simulation and scale them analytically to predict the performance of alternative designs.

However, GCStack and GCScaler are calibrated using Accel-Sim [7], whose microarchitectural model is rooted in the NVIDIA Tesla pipeline (2006) [14] with its hardware scoreboard-based dependency tracking, operand collector units, and Greedy-Then-Oldest (GTO) [15] scheduling policy. Recent reverse engineering studies have shown that modern NVIDIA GPUs differ fundamentally from this model [16]. They eliminate the operand collector in favor of compiler-guided dependency management encoded in per-instruction control bits and schedule warps with a Compiler-Guided Greedy-Then-Youngest (CGGTY) policy. Because GCScaler is calibrated with per-kernel cycles per instruction (CPI) stacks and per-instruction cache statistics collected on the simulator, these discrepancies propagate directly into the analytical model. Even where the legacy simulator’s cycle totals remain plausible through compensating errors, its stall decomposition and cache statistics misattribute the underlying behavior, degrading the baseline characterization and any scaled predictions derived from it.

In this paper, we investigate how the fidelity of the calibration source affects analytical GPU performance modeling. We develop *GCStack+*, which produces CPI stacks and cache statistics from a remodeled simulator of modern NVIDIA GPUs, and extend *GCScaler+* to leverage these improved calibration inputs for performance prediction. The framework further supports Blackwell-generation instruction sets and a protobuf-based trace format, and is validated against real hardware measurements on an NVIDIA RTX 5070 Blackwell GPU. The primary contributions of this work are as follows:

- We revisit the calibration process used by analytical GPU performance models and demonstrate that improving the quality of the calibration data leads to more accurate performance prediction. To this end, we develop *GCStack+*, which generates CPI stacks and per-instruction cache statistics using a remodeled simulator that also integrates CGGTY scheduling policy. By capturing these complex

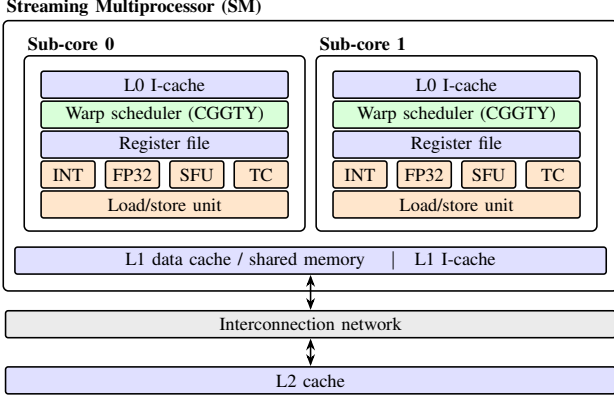


Fig. 1: High-level GPU architecture with the sub-core organization of a modern SM.

microarchitectural interactions, *GCStack+* enables the calibration of analytical models with execution behaviors representative of modern GPUs.

- We extend *GCScaler+* to leverage these calibration inputs for analytical performance prediction. The framework also supports Blackwell-generation ISA, a protobuf-based trace format, and a CGGTy scheduling model.
- We validate the proposed framework against an NVIDIA RTX 5070 across nine Rodinia benchmarks. *GCScaler+* reduces the geometric mean execution cycle prediction error from 18.2% to 12.1% while remaining $10.3\times$ faster than the simulator used during calibration.

II. BACKGROUND AND RELATED WORK

A. GPU Architecture

Figure 1 illustrates the high-level GPU architecture. Modern NVIDIA GPUs organize their compute resources around Streaming Multiprocessors (SMs), each containing multiple sub-cores that share an L1 data cache, shared memory, and an L1 instruction cache. Each sub-core maintains a private L0 instruction cache, a warp scheduler, a banked register file, and a set of functional units for integer, floating-point, special function, and tensor core operations. Load and store operations are handled by dedicated units that access the L1 data cache, the distributed L2 cache, and off-chip dynamic random-access memory (DRAM). Warps of 32 threads serve as the fundamental scheduling unit, and each sub-core manages multiple concurrent warps whose number is constrained by register and shared memory availability. When a kernel is launched, its threads are organized into thread blocks that are distributed across SMs based on available resources.

B. Cycle-Level GPU Simulators

GPGPU-Sim [17] models a GPU pipeline based on the NVIDIA Tesla architecture from 2006. Dependency tracking relies on hardware scoreboards, issued instructions are buffered in operand collector units until all source operands are retrieved, and the warp scheduler implements a GTO policy. Accel-Sim [7] extends GPGPU-Sim with trace-driven simulation support, sub-core modeling resembling NVIDIA Volta, and sectored

caches, but retains these fundamental Tesla-era assumptions. Reverse engineering of NVIDIA Ampere and Turing GPUs by Huerta et al. [16] has shown that modern sub-cores work differently. The compiler encodes per-instruction control bits (a stall counter for fixed-latency producers, a yield bit that releases the scheduler, and up to six dependence counters per warp for variable-latency operations), which eliminates hardware scoreboards and operand collectors entirely. The warp scheduler follows a CGGTy policy, preferring the same warp until its next instruction is not ready and then switching to the youngest ready warp. Huerta et al.’s remodeled simulator incorporating these findings achieves a mean absolute percentage error (MAPE) of 13.98% in execution cycles against NVIDIA RTX A6000 Ampere, an 18.24% improvement over Accel-Sim.

C. Analytical GPU Performance Models

Analytical models abstract the GPU microarchitecture into closed-form expressions to predict performance without cycle-level simulation, offering orders-of-magnitude speedups over timing simulation at the cost of accuracy. GPUMech [9] introduced GPU interval analysis, representing application behavior through a single representative warp and constructing CPI stacks by analytically scheduling that warp’s instruction stream. MDM [10] extended this approach with improved memory contention modeling. GCoM [11] further refined the framework by modeling memory and compute structural stalls and capturing load imbalance across the SM and sub-core hierarchy. Despite these advances, analytical models rely on coarse-grained interval-level stall classification with fixed priority between stall events, which exaggerates memory-side stalls and overlooks concurrent core-side stalls.

D. Hybrid Approaches

Swift-Sim [18] performs cycle-accurate simulation only for the components of interest and relies on analytical models for the remaining hardware components. *GCStack* and *GCScaler* [12], [13] bridge the gap between the accuracy of cycle-level simulation and the efficiency of analytical modeling. *GCStack* is a fine-grained cycle-accounting mechanism built on top of Accel-Sim that constructs CPI stacks by distributing each stall cycle across all warp slots. This approach captures concurrently occurring stall events rather than attributing each cycle to a single event according to a priority order. *GCScaler* builds on the baseline CPI stack generated by *GCStack* and uses scale factors derived from an interval analysis model to predict CPI stacks for alternative GPU designs without requiring additional cycle-level simulations. It further improves prediction accuracy through idle subclassification, analytical scaling of synchronization and empty-warp-slot stalls, and calibrated representative-warp selection.

Both *GCStack* and *GCScaler* were developed on top of Accel-Sim’s outdated pipeline model. Our work ports the cycle-accounting mechanism to the remodeled simulator introduced by Huerta et al., resulting in *GCStack+*. We show that the resulting CPI stacks provide a more faithful calibration baseline for the analytical model, which we extend as *GCScaler+*.

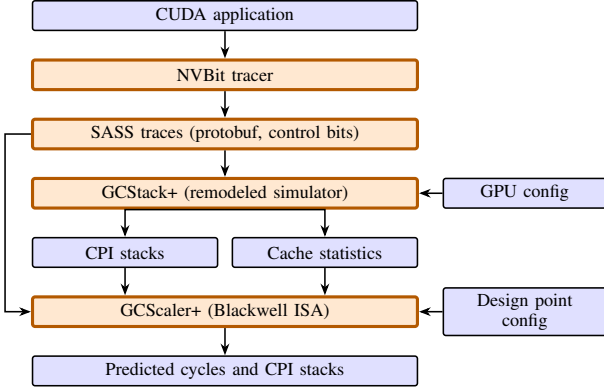


Fig. 2: Workflow overview of GCStack+ and GCScaler+, with updated components highlighted in orange.

III. IMPLEMENTATION

A. Overview

Figure 2 gives an overview of GCStack+ and GCScaler+. The application runs once on real hardware under an NVBit-based [19] tracer that instruments every kernel at the Streaming AS-Sembly (SASS) level and records per-warp instruction streams with opcodes, register operands, and memory addresses. The legacy tracer emits the text-based format of Accel-Sim. The enhanced tracer replaces it with a structured protobuf format that also captures compiler control bits.

GCStack consumes the traces together with a hardware configuration and performs a cycle-level simulation of the GPU. While simulating, its fine-grained cycle accounting inspects every scheduler cycle and distributes it across all warp slots according to the condition that blocks each warp, so concurrently occurring stall events are all captured. Aggregated over a kernel, this yields the per-kernel CPI stack with base, compute and memory data, compute and memory structural, synchronization, control, and subclassified idle components. The simulation additionally records per-instruction cache statistics, namely the L1 and L2 hit and miss counts and the accessed addresses of every memory instruction of every warp. These two artifacts calibrate the analytical model. GCStack performs this accounting on the Tesla-era Accel-Sim pipeline with hardware scoreboards and operand collectors, while GCStack+ performs it on the remodeled simulator [16].

GCScaler decodes the same traces a second time. Instead of simulating the memory hierarchy, it replays the recorded cache statistics. For each kernel it selects a representative warp by running the interval model for every candidate warp and keeping the one whose modeled CPI stack is most similar to the simulator’s CPI stack for that kernel. The representative warp’s instruction stream is then split into intervals, which are sequences of issued instructions terminated by a stall. From this interval profile a sub-core model derives base cycles, data stalls after latency hiding (Section III-D), and structural stalls from functional unit and memory contention. The sub-core results are composed across the sub-cores and SMs of the configured GPU, idle cycles are added for load imbalance and

inactive SMs together with the per-kernel launch overhead, and the outcome is the predicted cycle count and CPI stack of each kernel. With fixed traces, CPI stacks, and cache statistics, evaluating an alternative design only requires changing the hardware configuration and rerunning the fast analytical model.

GCScaler+ adds Blackwell instruction decoding, the protobuf trace format, and the Greedy-Then-Youngest (GTy) scheduling policy of modern hardware. In the following sections, we describe the main improvements in more detail.

B. Adapting Stall Accounting to Compiler-Guided Dependency Tracking

GCStack’s stall accounting was designed for the Accel-Sim pipeline. Its lazy structural stall classification, for example, compensates for the operand collector’s back-pressure by deferring the memory-side versus compute-side distinction to the execute stage. On a simulator without operand collectors this artifact disappears, the split-path execution model produces different stall patterns, and the remodeled memory pipeline alters per-instruction cache hit/miss rates that GCScaler consumes.

Our first contribution is *GCStack+*, a port of GCStack’s fine-grained stall cycle accounting and per-instruction cache statistics collection to the remodeled simulator [16]. GCStack+ tracks the same per-kernel CPI components as GCStack, but derives warp readiness from the compiler-set stall counters and dependence counter masks of the remodeled pipeline rather than from hardware scoreboards. We further adapt the cache statistics collection to use instruction-level memory indexing consistent between the simulator and the analytical model’s trace-driven cache replay, so that the recorded statistics of either simulator generation can be replayed deterministically. Our second contribution is *GCScaler+*, the extension of GCScaler described in the remainder of this section, which consumes the GCStack+ inputs.

C. Enabling Blackwell ISA and Enhanced Trace Support

To extend coverage to the latest NVIDIA architectures, we add support for Blackwell-generation SASS instruction sets in both the simulator and the analytical model. This includes mapping Blackwell opcodes to their corresponding functional units, covering integer, floating-point, special function, tensor core and branch operations.

We also integrate an enhanced protobuf-based trace format that replaces the legacy text-based format used by Accel-Sim. The protobuf schema defines structured kernel and threadblock messages that capture grid and block dimensions, register and shared memory configuration, and per-warp instruction traces with full operand information. The trace parser supports both formats, detecting the trace type at initialization and dispatching to the appropriate parsing backend, which allows GCScaler+ to consume inputs from both simulator generations.

D. Compiler-Guided Greedy-Then-Youngest Scheduling

The remodeled hardware schedules warps with the CGGTy policy instead of the GTO policy that GCScaler inherited from GCoM. GCScaler+ therefore adds CGGTy to the latency-hiding model as a configurable scheduling policy and uses it by

default. The model estimates how many instructions the non-greedy warps issue while the greedy warp stalls for C_{stall} cycles. Under GTO this count is bounded by a capped per-interval issue probability

$$I_{\text{stall}}^{\text{GTO}} = \min(C_{\text{stall}} \cdot p_{\text{issue}}, 1) \cdot W_{\text{oth}} \cdot \bar{I}_{\text{intv}} \quad (1)$$

where W_{oth} is the number of non-greedy warps per sub-core, $\bar{I}_{\text{intv}}$ the average instructions per interval of the representative warp, and $p_{\text{issue}} = \sum I_{\text{intv}} / (\sum I_{\text{intv}} + \sum C_{\text{stall}})$ the global issue probability. Under CGGTy the scheduler switches to the youngest ready warp, whose readiness the model represents directly by the issue probability

$$I_{\text{stall}}^{\text{GTy}} = W_{\text{oth}} \cdot p_{\text{issue}} \quad (2)$$

which drops the per-interval instruction factor and therefore hides less latency per stall. The sums in p_{issue} run over all intervals of the representative warp, so it is a static per-kernel average and only C_{stall} varies between intervals. The residual data stall $C_{\text{data}} = \max(C_{\text{stall}} - I_{\text{stall}}/r_{\text{issue}}, 0)$ with sub-core issue rate r_{issue} is shared by both policies. We evaluate the effect of CGGTy on accuracy in Section V-A.

E. Scaling for Alternative GPU Designs

GCScaler’s original scaling framework predicted performance for alternative GPU designs by applying scale factors to baseline CPI stacks collected from a single reference simulation. GCScaler+ adapts this framework to operate on the CPI stacks produced by GCStack+. The hardware configuration specifies SM and sub-core counts, functional unit latencies, cache hierarchy parameters, network-on-chip (NoC) and DRAM bandwidth, and the per-kernel launch overhead. Changing these parameters predicts the CPI stacks of alternative designs without repeating cycle-level simulations.

IV. EXPERIMENTAL SETUP

A. Hardware Platform and Trace Collection

We validate the framework on an NVIDIA RTX 5070. Table I lists the hardware configuration and calibrated analytical model parameters. All simulation and analytical modeling are performed on a 24-core Intel Core i9-13900K host with 128 GB of memory. We collect SASS instruction traces using an NVBit [19] based tracer that records per-warp instruction streams including opcodes, operands, memory addresses, and control bits. Each application is traced once in the legacy text format consumed by Accel-Sim/GCStack and once in the enhanced protobuf format consumed by GCStack+. Per-kernel execution cycles and warp stall statistics are collected using NVIDIA Nsight Compute (NCU).

B. Simulation Flows

We compare four flows against the hardware measurements. *GCStack* is the original Accel-Sim-based implementation. *GCStack+* is our port to the remodeled simulator incorporating compiler-guided dependency management and the absence of operand collectors. Each simulator produces per-kernel CPI stacks and per-instruction cache statistics, which are fed into the

TABLE I: NVIDIA RTX 5070 hardware parameters and calibrated model parameters. Memory latencies, bandwidths and the core clock are taken from the configs of the remodeled simulator [16].

Parameter	Value
SMs	48 SMs @ 2.58 GHz, four sub-cores each, 1536 threads/SM, 32 TBs/SM, 65K registers/SM
Warp scheduling	GTO and CGGTy (Section III-D)
L1 data cache	128 KB/ SM, unified, sector, write-through, 39-cycle latency
Shared memory latency	29 cycles
L2 cache	48 MB, 24-way, write-back, eight sub-partitions/ channel, 301-cycle latency
DRAM	448 GB/s, 16 channels, 255-cycle latency
Execution latencies	INT and FP32 2, SFU 21, FP64 64 cycles
Kernel launch overhead	4,000 cycles

TABLE II: Benchmarks used for experimental evaluation.

Benchmark	Configuration	Kernels
B+Tree	1M records	2
Backprop	65,536 input nodes	2
BFS	1M nodes	24
Gaussian	4 × 4 matrix	6
Hotspot	1024 ² grid	1
Hotspot3D	512 ² × 8 grid, 100 iter.	100
LUD	512 ² matrix	94
NN	default input	1
NW	2048 ² , penalty 10	255

interval analysis model, yielding *GCScaler* (GCStack inputs) and *GCScaler+* (GCStack+ inputs). The analytical model uses the parameters of Table I, the cache hierarchy, memory bandwidth constraints, and a constant per-kernel launch overhead.

C. Benchmarks and Metrics

Table II summarizes the Rodinia 3.1 benchmarks [20] used in our evaluation. The benchmark suite spans memory-bound, compute-bound, and mixed workloads with diverse memory access patterns, branch divergence, and kernel counts.

We report three metrics. *Execution cycle prediction error* is the per-benchmark mean relative error in predicted kernel cycles compared to hardware measurements, summarized by its geometric mean across benchmarks. *CPI stack accuracy* compares the per-kernel CPI breakdown (base, compute data/structural, memory data/structural, control, idle) produced by the analytical model against reference CPI stacks derived from NCU warp stall statistics. *Runtime* is the wall-clock time of each flow over the full benchmark set.

V. RESULTS

A. Execution Cycle Prediction Accuracy

Figure 3 shows the relative error in predicted execution cycles for each benchmark and flow. Between the two cycle-level simulators, the remodeled pipeline improves the geometric mean error from 17.4% to 10.2%. GCStack+ strongly improves the benchmarks with the largest legacy errors, most visibly Gaussian (55.7% to 16.8%) and Nearest Neighbor (30.7% to 0.3%), but regresses B+Tree and Hotspot3D. The analytical

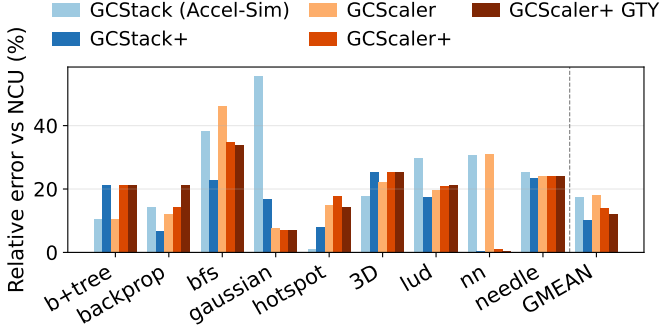


Fig. 3: Relative error in execution cycles compared to RTX 5070 hardware measurements.

model extracts a further benefit from the modern inputs, GCScaler+ reaches a geometric mean error of 12.1% with its default CGGTy scheduling policy and 13.9% with GTO, versus 18.2% for GCScaler. The interval model’s calibrated representative warp selection and explicit kernel launch overhead compensate for a large part of the simulator’s residual biases.

The two simulator regressions share a common signature. GCStack+ underestimates the cycles of every B+Tree and Hotspot3D kernel, by 21% and 25% on average, pointing to overly aggressive latency hiding in the remodeled pipeline. The analytical model follows its calibration source and reproduces these two regressions almost exactly, but it does not inherit every simulator error. On Gaussian GCScaler+ reaches 6.9% where GCStack+ is at 16.8%.

Exchanging the two inputs separately isolates where the gain originates. Starting from the legacy input set at 18.2%, the modern cache statistics alone improve the geometric mean error to 17.0%, while the modern CPI stacks alone reach 13.5%. The full GCStack+ input set lands at 13.9%. The CPI stacks thus deliver the larger share of the improvement. The CPI-stack targets also carry the simulator’s latency-hiding bias into the representative warp selection. On B+Tree and Hotspot3D the error follows the CPI-stack generation regardless of the cache statistics, which marks the CPI-stack side of GCStack+ as the component to recalibrate.

B. Runtime

Table III reports the wall-clock time of each flow over all nine benchmarks, together with its accuracy. GCStack+ outpaces legacy GCStack because of its OpenMP-parallel SM simulation, and GCScaler+ is faster than GCScaler because the protobuf traces parse faster than the legacy text traces. GCScaler+ evaluates the full suite in under eight minutes, 10.3 $\times$ faster than the GCStack+ simulation it is calibrated by and 40.4 $\times$ faster than legacy GCStack. It requires one baseline simulation, and its speed advantage then applies to every subsequent design point.

C. Trace Footprint

Table IV compares the two trace formats. Uncompressed, the protobuf traces are 2.7 $\times$ smaller than the text traces. The

TABLE III: Total wall-clock time and accuracy over the nine Rodinia benchmarks. Speedup is relative to GCStack.

Flow	Total runtime	Speedup	GMEAN error
GCStack	19,238 s	1 $\times$	17.4%
GCStack+	4,914 s	3.9 $\times$	10.2%
GCScaler	543 s	35.4 $\times$	18.2%
GCScaler+ GTO	477 s	40.4 $\times$	13.9%
GCScaler+ GTY	477 s	40.4 $\times$	12.1%

TABLE IV: Trace footprint over the nine benchmarks.

	Accel-Sim text	Protobuf
Traces, uncompressed	19.4 GB	7.2 GB
Traces, as stored	2.0 GB	7.2 GB
Traces, single xz archive	151 MB	152 MB

formats are stored differently, as Accel-Sim compresses each kernel into a separate xz file while the protobuf tracer writes one uncompressed file per threadblock, which leaves the text traces smaller on disk. Compressing both into a single xz archive brings them within 1% of each other, so the binary encoding mainly removes redundancy that a compressor removes as well. The benefit of the protobuf format is therefore parsing throughput since the traces are read without decompression and without text conversion.

D. CPI Stack Accuracy

To understand *where* the analytical model deviates from hardware behavior, we compare the CPI stacks produced by GCScaler and GCScaler+ against stacks derived from NCU counters. The hardware stack takes issued and idle cycles from exact counters and apportions the remaining stall pool to the model’s categories using the warp stall reason ratios. Figure 4 shows the comparison, with each bar scaled by its predicted-to-measured cycle ratio. That ratio compares summed cycles, so per-kernel errors of opposite sign can cancel and a stack can appear more accurate than Figure 3 suggests. B+Tree illustrates this. GCScaler overestimates its first kernel by 12.3% and underestimates its second by 8.6%, so its total matches the hardware almost exactly while the mean per-kernel error stays at 10.4%.

Both variants underestimate total cycles by a similar margin (geometric mean cycle ratio 0.88 for GCScaler versus 0.85 for GCScaler+), so the accuracy gain of GCScaler+ stems from correcting per-benchmark biases. The stack composition improves with the modern inputs. The idle fraction, which dominates the small-kernel Rodinia workloads, deviates from hardware by 4.9 percentage points on average for GCScaler but only 1.9 for GCScaler+, and the compute structural fraction improves from 6.7 to 5.0 percentage points. The compute and memory data stalls remain the largest composition error for both variants (8.6 to 10.6 percentage points each).

E. Impact of Kernel Launch Latency

Interval analysis models a kernel from its instruction trace, so the latency of launching the kernel is invisible to it. A null-kernel microbenchmark on the RTX 5070 measures about 4,000

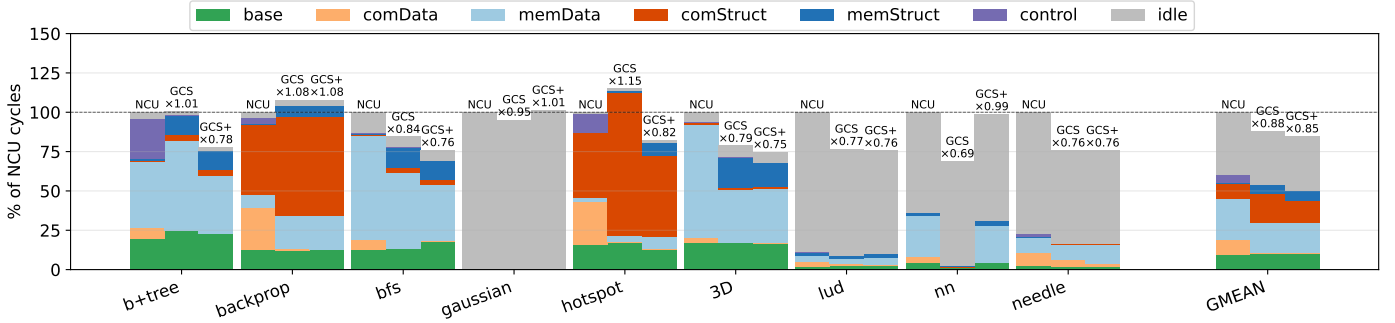


Fig. 4: CPI stacks of GCScaler (GCS) and GCScaler+ (GCS+), scaled by the predicted-to-measured cycle ratio. The NCU bar (100%) is the hardware reference. The GMEAN group shows the geometric mean cycle ratio.

cycles per launch for back-to-back launches in a single stream. The analytical model therefore adds a constant launch overhead to every predicted kernel, set to 4,000 cycles in Table I.

To quantify the effect of this constant we repeat the accuracy experiment for GCScaler+ with the launch overhead set to zero. The geometric mean error grows from 13.9% to 30.7%. The degradation occurs in benchmarks that launch many short kernels, where the fixed launch cost makes up a large fraction of every kernel’s measured cycles. Gaussian’s kernels run about 4,900 cycles each and its error grows from 6.9% to 74.1%, and NW and LUD more than double their errors.

F. Design Space Exploration

Finally, we exercise the scaling path itself. GCScaler+ predicts alternative GPU designs from the *baseline* inputs alone, validated against GCStack+ simulations of the same designs. We halve and double the SM count and the DRAM bandwidth for two benchmarks, and Figure 5 compares the speedup over the baseline design that both sides observe. Across the eight design points the predicted cycles deviate from the simulated ones by 9.3% in the geometric mean. The simulation shows both benchmarks to be largely insensitive to DRAM bandwidth, and the model reproduces this. It also captures Backprop’s slowdown at half the SM count. The misses concentrate on the SM count itself. The model underpredicts the gain from doubling it on both benchmarks, most strongly on B+Tree (1.15 $\times$ predicted vs 1.87 $\times$ simulated at double the count). The cause is the memory contention model, whose queueing delay grows linearly with the number of active SMs irrespective of whether the interconnect is saturated, so the added SM parallelism is partly cancelled by additional modeled queueing. The SM-scaling path therefore deserves further calibration.

VI. CONCLUSION

This work addresses the calibration gap in hybrid GPU performance modeling caused by relying on simulators with outdated microarchitectural assumptions. We presented GCStack+, which integrates GCStack’s fine-grained cycle accounting into a remodeled simulator that captures the execution model of modern NVIDIA GPUs, and GCScaler+, an extended interval analysis model that leverages the resulting CPI stacks and cache statistics while adding support for Blackwell

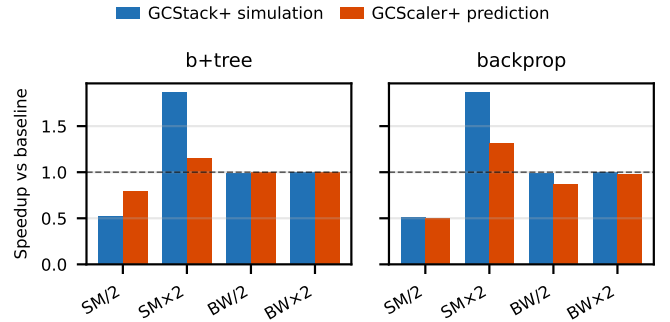


Fig. 5: Design space exploration: speedup over the baseline when halving or doubling the SM count and DRAM bandwidth, GCStack+ simulation of each design versus GCScaler+ prediction from baseline inputs.

ISA, a protobuf-based trace format, and a GTY scheduling model. On an NVIDIA RTX 5070, GCScaler+ achieves a geometric mean execution cycle prediction error of 12.1% with its GTY scheduling model, improving on GCScaler (18.2%), while evaluating the full suite 10.3 $\times$ faster than GCStack+. Comparison with NCU profiling shows that both calibration inputs underestimate the overall CPI stack by a similar margin, while GCStack+ more closely matches the hardware in the idle and compute structural categories. These results demonstrate that the quality of the calibration inputs is a crucial factor in analytical GPU performance modeling, and that modeling kernel launch overhead matters for workloads composed of many short kernels. As future work, we plan to improve the CPI stack category mapping to hardware profiling and to investigate cross-architecture scaling across GPU generations.

ACKNOWLEDGMENT

In accordance with the IEEE policy on AI-generated content, Anthropic’s Claude Opus 4.8 was utilized to improve grammar and overall readability.

This research is partially supported by the following grants: NSF-MeitY Grant (Y-20/5/2024-R&D-E (e-file: 3149156)),

REFERENCES

- [1] A. Ilievski, V. Zdravski, and M. Gusev, "How cuda powers the machine learning revolution," in *2018 26th Telecommunications Forum (TELFOR)*, pp. 420–425, IEEE, 2018.
- [2] M. Li, Z. Bi, T. Wang, Y. Wen, Q. Niu, X. Song, Z. Jiang, J. Liu, B. Peng, S. Zhang, *et al.*, "Deep learning and machine learning with gpgpu and cuda: Unlocking the power of parallel computing," *arXiv preprint arXiv:2410.05686*, 2024.
- [3] J. Lew, D. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers, *et al.*, "Analyzing machine learning workloads using a detailed gpu simulator," *arXiv preprint arXiv:1811.08933*, 2018.
- [4] W. Liu, B. Schmidt, G. Voss, and W. Müller-Wittig, "Accelerating molecular dynamics simulations using graphics processing units with cuda," *Computer physics communications*, vol. 179, no. 9, pp. 634–641, 2008.
- [5] L. Nylons, M. Harris, and J. Prins, "Fast n-body simulation with cuda," *GPU gems*, vol. 3, pp. 62–66, 2007.
- [6] H. Nam, J. H. Lee, Y. Kim, M. Zhang, J. Kim, and B. Burgstaller, "Lpgsim: A lightweight parallel gpu simulator maximizing speed with trustworthy simulation," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 10, May 2026.
- [7] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 473–486, IEEE, 2020.
- [8] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao, *et al.*, "Mgpusim: Enabling multi-gpu performance modeling and optimization," in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 197–209, 2019.
- [9] J.-C. Huang, J. H. Lee, H. Kim, and H.-H. S. Lee, "Gpumech: Gpu performance modeling technique based on interval analysis," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 268–279, IEEE, 2014.
- [10] L. Wang, M. Jahre, A. Adileho, and L. Eeckhout, "Mdm: The gpu memory divergence model," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1009–1021, IEEE, 2020.
- [11] J. Lee, Y. Ha, S. Lee, J. Woo, J. Lee, H. Jang, and Y. Kim, "Gcom: a detailed gpu core model for accurate analytical modeling of modern gpus," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 424–436, 2022.
- [12] H. Cha, S. Lee, Y. Ha, H. Jang, J. Kim, and Y. Kim, "Gcstack: A gpu cycle accounting mechanism for providing accurate insight into gpu performance," *IEEE Computer Architecture Letters*, vol. 23, no. 2, pp. 235–238, 2024.
- [13] H. Cha, S. Lee, J. Lee, Y. Ha, J. Kim, and Y. Kim, "Gcstack+ gscaler: Fast and accurate gpu performance analyses using fine-grained stall cycle accounting and interval analysis," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 1509–1523, 2025.
- [14] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE micro*, vol. 28, no. 2, pp. 39–55, 2008.
- [15] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 72–83, IEEE, 2012.
- [16] R. Huerta, M. A. Shoushtary, J.-L. Cruz, and A. Gonzalez, "Dissecting and modeling the architecture of modern gpu cores," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, pp. 369–384, 2025.
- [17] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE international symposium on performance analysis of systems and software*, pp. 163–174, IEEE, 2009.
- [18] X. Xu, Y. Lv, L. Wang, L. Xiao, M. Han, R. Shen, and J. Wang, "Swiftsim: A modular and hybrid gpu architecture simulation framework," in *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7, 2025.
- [19] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 372–383, 2019.
- [20] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*, pp. 44–54, Ieee, 2009.